Journal of Big Data

RESEARCH Open Access

A better entity detection of question for knowledge graph question answering through extracting position-based patterns

Mohammad Yani[1,2], Adila Alfa Krisnadhi[1*] and Indra Budi[1]

*Correspondence: adila@cs.ui.ac.id

[1] Faculty of Computer Science, Universitas Indonesia, Depok, Indonesia
[2] Informatics Department, Politeknik Negeri Indramayu, Indramayu, Indonesia

Abstract

Entity detection task on knowledge graph question answering systems has been studied well on simple questions. However, the task is still challenging on complex questions. It is due to a complex question is composed of more than one fact or triple. This paper proposes a method to detect entities and their position on triples mentioned in a question. Unlike existing approaches that only focus on detecting the entity name, our method can determine in which triple an entity is located. Furthermore, our approach can also define if an entity is a head or a tail of a triple mentioned in a question. We tested our approach to SimpleQuestions, LC-QuAD 2.0, and QALD series benchmarks. The experiment result demonstrates that our model outperforms the previous works on SimpleQuestions and QALD series datasets. 99.15% accuracy and 96.15% accuracy on average, respectively. Our model can also improve entity detection performance on LC-QuAD 2.0 with a merged dataset, namely, 97.4% accuracy. This paper also presents Wikidata QALD series version that is helpful for researchers to assess the knowledge graph question answering system they develop.

Keywords: Entity detection, Entity recognition, Knowledge graph question answering, Complex question, Question pattern

Introduction

Knowledge graph question answering (KGQA) systems enable users to retrieve data from a knowledge graph (KG) using a natural language question (NLQ). In KG, data is modeled as a collection of triples that conform to a schema. Those triples are expressed in a particular language, such as Resource Description Framework (RDF) [1].

One approach taken by many KGQA systems is through the so-called semantic-based parsing [2]. In this approach, a given NLQ is translated into an equivalent logical query appropriate for the language representing the KG. For example, if the KG is expressed using RDF, then the NLQ is translated into a SPARQL query as SPARQL is the standard query language for RDF data. This query can then be used to obtain an answer from the KG directly.

Such a translation is non-trivial as it requires the system to accomplish several tasks. At the start, it needs to detect the occurrences (i.e., mentions) of entities and relations in the given NLQ. Then, once the entities and relations are known, it needs to link them to the appropriate entities and relations in the KG. Finally, it has to form a correct query using the linked entities and relations. Thus, a good translation should yield a query that not only captures the intent of the input NLQ, but is also correctly composed of entities and relations in the KG that accurately correspond to entities and relations in the input NLQ.

Yani and Krisnadhi [3] surveyed a number of challenges and approaches in developing simple KGQA systems, i.e., KGQA systems that are capable of answering simple questions, which correspond to queries containing only a single triple pattern. They categorized the challenges according to the tasks the systems need to accomplish, namely, entity detection, entity linking, relation prediction, relation linking, answer matching (by query construction or embedding-based methods), and subgraph selection. KGQA systems are typically composed of solutions to these tasks arranged sequentially as a pipeline. Obviously, the performance of the earliest task, i.e., entity detection, may significantly influence the performance of the subsequent tasks. Accordingly, addressing the entity detection task is a key step to realize a powerful KGQA system.

Entity detection aims to determine which N-gram mentions in the given NLQ actually represent entities. To accomplish this task, one sometimes takes an "easy" approach by using Named Entity Recognition (NER) libraries, such as Stanza.[1] The output of such a NER tool is a labeling of all tokens occurring in the input NLQ. One then proceeds by classifying the N-grams from those tokens into entities resulting in a set of detected entities. Based on this set, one then searches for *all* of these entities in the subsequent entity linking step to obtain matching entities in the KG.

However, there are three issues in this approach. First, the input NLQ may have more than one named entities even when it actually corresponds to a single factoid fact. For example, the question "Where in the United States was John Morris Russell born?" from the SimpleQuestions dataset [4] contains two entities detected by NER, namely "the United States" and "John Morris Russell". If one stops at entity detection only, then this is a non-issue. But in a KGQA system, one wishes to use the detected entities in the subsequent entity linking and query construction phases. Meanwhile, as exemplified by the SimpleQuestions dataset, such a question may correspond to a query that contains the occurrence of only a single entity. Thus, if NER detects two entities, one needs to choose which of the two is the appropriate one to pass to the subsequent phases.

Second, the input NLQ may contain an entity, but NER may not recognize it. Consider the question "What player plays the position midfielder?" from the SimpleQuestions dataset. In this question, NER returns an empty set of named entities. Thus, the system needs to find an alternative way, e.g., using an additional corpus, to extract the entities from the question before linking them to KG entities.

Third, a complex NLQ translates to a complex query composed of a set of simpler queries involving many functions and operators [5]. For a KG in RDF, a complex question corresponds to a SPARQL query with more than one triples. This leads to three

[1] Stanza is based on StanfordCoreNLP - online version is at http://stanza.run/.

```
SELECT ?value1 ?value2 WHERE {
    wd:Q206191 p:P40 ?s .
    ?s ps:P40 wd:Q4667661 .
    ?s pq:P22 ?value1 .
    ?s pq:P569 ?value2 .
}
```

Fig. 1 Ground truth SPARQL query for the question "What is the birthday of Abigail Adams who has a child named Abigail Adams Smith?" from the Lc-QuAD 2.0 dataset

challenges: how do we determine the required triple number? In which triple do entities exist? Do those entities exist as a head or a tail in a triple?

For example, the question "What is the birthday of Abigail Adams who has a child named Abigail Adams Smith?" from LC-QuAD 2.0 dataset [6] corresponds to a ground truth query with four triple patterns as depicted in Fig. 1. In the query, the entity Abigail Adams appears as the head of the first triple, while the entity Abigail Adams Smith appears as the tail of the second triple. In addition, no entity appears in the remaining two triples. Given an NLQ, the number of triples in the corresponding query is unknown beforehand. We also do not know both in which triple(s) do entities mentioned in the question appear and whether they appear at the head or tail position.

The three aforementioned issues motivate us in this paper to focus on the entity detection task. Specifically, given an NLQ, we intend to detect the occurrences of entities—named or otherwise—in the NLQ, and simultaneously determines in which triples do these entities appear and whether these entities appear as the head or the tail of the triples. Note that this goes beyond detecting whether an N-gram mention in the NLQ is an entity as we also indirectly predict how the KG query corresponding to the NLQ could be constructed.

Existing works on entity detection mainly addressed the multiple-named-entities issue (the first issue), and to a certain extent, the undetected-entity issue (the second issue). The first issue is tackled by performing n-gram and sequence labeling approach [7–12]. The second issue is tackled by using a dictionary that contains pairs of mentions and entities in the KG [13, 14]. Embedding-based approaches are also used to address this issue [15, 16].

Meanwhile, the third issue is typically not tackled during entity detection phase. For example, Evseev and Arkhipov [11] employs SPARQL templates created at the query construction phase. In this work, a fixed number of SPARQL templates is manually formulated, each of which is intended to represent a certain type of query. A classifier is then trained to categorize input NLQs into one of those templates. However, the problem with this approach is that one still needs to fill the slots in the templates with the appropriate entity from the input NLQ to derive the correct answer. Evseev and Arkhipov's solution requires ranking the matching score of *all* possible permutations of entities in the NLQ against the ground truth queries of each template.

```
SELECT ?ans_1 ?ans_2 WHERE {
  wd:Q106942 wdt:P25 ?ans_1 .
  wd:Q106942 wdt:P26 ?ans_2 .
}
```

Fig. 2 Ground truth SPARQL query for the question "Who is the mother and husband of Candice Bergen?" from the LC-QuAD 2.0 dataset

In contrast, our work addresses all three issues at once during entity detection by employing the so-called *position-based patterns*. The premise of this idea is that each NLQ can be viewed as an instance of some question type. Instead of manually formulating the question types, we make use of ground truth triples in the SPARQL queries that correspond to NLQs from the training dataset to generate those question types. More precisely, given the ground truth query of an NLQ in the dataset, we can obtain all entities appearing in any triple patterns of the query. Then, for each of those entities, we encode three information in a position-based pattern: (i) the (index of) the triple pattern of the query (first, second, etc.) in which the entity appears, (ii) whether the entity appear in the head or tail position, and (iii) the token position of the entity in the NLQ. This token position is obtained by a simple string matching between the entity's label according to the KG and the N-gram mentions in the NLQ. Thus, from the ground truth query of an NLQ, we obtain a *set of such position-based patterns*, which can be used as a representation of the *question type* of the NLQ. The solution to entity detection task is then derived by training a classifier that categorizes the input NLQs into a position-based pattern set that represents the appropriate question type.

As an example, consider the NLQ "Who is the mother and husband of Candice Bergen?" from the LC-QuAD 2.0 dataset. This NLQ corresponds to the ground truth SPARQL query as listed in Fig. 2 where wd:Q106942 is the Wikidata instance for "Candice Bergen", wdt:P25 is the Wikidata property "mother", and wdt:P26 is the Wikidata property "spouse". Position-based patterns for the above NLQ are 0:($\texttt{head}, [7, 8]$) and 1:($\texttt{head}, [7, 8]$) indicating that the entity formed by the 7th and 8th token of the NLQ is a head entity of both the 0th and 1st triple pattern in the SPARQL query. Note that if the NLQ contains other N-gram mentions that could be considered an entity, but they do not appear in the ground truth query, then those N-gram mentions are ignored.

The two position-based patterns above form a position-based pattern set for the aforementioned NLQ. One such position-based pattern set represents one particular question type to which many NLQs may be belong. Now, if an arbitrary input question is correctly classified by the trained classifier into that question type, then in the subsequent entity linking and query construction steps, one can simply extract the entity from the question and match it with a KG entity, and then construct a query containing a single triple pattern with the matched KG entity appearing as a subject.

The encoding above captures all necessary information to solve the third issue we discussed earlier. Meanwhile, multiple named entities in the input NLQ are taken care of since position-based patterns capture the token position of the entities that are actually needed for the query associated with the NLQ. Furthermore, the undetected entity issue is handled because position-based patterns rely on entities that actually occur in the triple patterns of the query regardless whether or not they are named entities detectable by NER.

On the other hand, we do *not* encode the entity label or identifier in the position-based patterns. Hence, different questions with completely different set of entities may generate the same set of position-based patterns if their ground truth queries have the same form and the N-gram length of their entity labels is similar. In our formulation, these questions belong to the same question type.

Note that we do not address the challenges concerning the entity linking task, which follows immediately after entity detection. In particular, we do not focus on solving the ambiguity challenge. This refers to the situation in which the same N-gram mention may have multiple meanings and thus can refer to different entities in the KG [5, 17]. This challenge, however, is out of scope for this paper because such a situation does not occur during entity detection task.

Contribution

As the main contribution, we present a solution to entity detection using position-based patterns that allows us to bypass the need to employ NER in the pipeline. More precisely, we directly train a transformer model that recognizes the token-based positions of some or all entities that appear in the input NLQ and whether the recognized entities are a subject (head) or an object (tail) of some triples in the KG. The target labels for this classification problem are generated based on the dataset's corresponding ground truth triple patterns.

Although this paper only proposes a solution to the entity detection task, the performance of entity detection can significantly influence the performance of subsequent tasks. Furthermore, the previous work such as Türe and Jojic [18] with a hybrid technique used; namely, GRU, Bi-GRU, LSTM, and Bi-LSTM can contribute to achieving the best performance of end-to-end performance on simple KGQA systems. In this paper, we demonstrate that our proposed approach can improve the downstream task of entity linking empirically through experiments with Falcon 2.0 [19], which is a tool for entity and relation linking over Wikidata.

In addition to the technical contribution of position-based patterns, we also introduce an expansion to benchmark datasets for the challenges in the Question Answering over Linked Data (QALD) series [20–28], which is a series of benchmarks for evaluating KGQA systems. Specifically, we introduce a Wikidata-version of the benchmark datasets for 8 of 9 QALD challenges. An exception is QALD-7 whose dataset already uses Wikidata[2] as an underlying KG. Note that all nine QALD challenges already use DBPedia[3] as an underlying KG. Since both DBPedia and Wikidata are originated from Wikipedia corpus, our Wikidata version is obtained by translating existing DBPedia queries in the benchmark datasets into Wikidata queries.

The rest of this paper is structured as follows. The Related Works section overviews previous approaches that address and evaluate the entity detection task. The Proposed

[2] https://wikidata.org/.

[3] https://www.dbpedia.org.

Method section presents our proposed approach to address entity detection tasks. This section also explains a procedure for translating DBPedia based QALD query into Wikidata based QALD query, which allows us to expand the QALD series datasets. In the Experimental Setup and Results section, we present our experimental setup and detail the evaluation we conducted over our system. For the latter, we specifically compare our results with four state-of-the-art works that explicitly assess the performance of entity detection tasks. The final section concludes our work and introduces the future work.

Related works

There are two kinds of approaches in answering questions over knowledge graphs, namely, SPARQL query-based and embedding-based approaches. SPARQL query-based approaches first extract entities and relations of the input question. Then, from the extracted entities and relations, it constructs the SPARQL query. The work by Song et al. [29] and Zou et al. [30] belong to this type of approaches. Meanwhile, the embedding-based approaches employ neural embedding methods to transform the extracted terms and structures as well as the KG into (numerical) vectors in a vector space. The answer to the input question is then obtained through the use of vector operations. Huang et al. [31] and Bordes et al. [4] proposed KGQA systems that belong to the embedding-based approaches.

In SPARQL query-based approaches, particularly those using NER, entity detection faces two main issues: multiple named entities in factoid questions and entities undetected by NER. Dai et al. [7] addressed the first issue by finding a pair of a named entities with a corresponding relation in the question. A named entity linked by the relation is assumed as the entity mentioned in the question. He and Golub [8] performed a detection of multiple named entities in a question by finding the closest similarity score between the KG and the input question. However, finding all possible relations that correspond to an entity is computationally expensive.

Chao and Li [9] used unigram, bigram, and trigram fragments of the text representation of entities to be matched to entities in the input question. Their approach takes the longest n-grams as entity candidates to solve the issue of entities undetected by NER. However, this approach has not solved the first issue, i.e., multiple named entities in the input question.

Lukovnivov et al. [10] employed a BERT model to detect the span s in the input question q that mentions an entity. Here, a pre-trained BERT model contains a rich amount of information as it has been trained from the underlying large corpus. This allows BERT to identify the span in the question that mentions an entity but at the risk of having a large number of candidate entities to match. This introduces a difficult ambiguity problem in the subsequent entity linking task.

Evseev and Arkhipov [11] proposed a solution to entity detection by training a BERT-based sequence labeling model on a corpus of questions. Unlike our multiclass classification model, this is a multilabel classification model where the input are the tokens of the questions and the target are taken from entity labels in the KG that may match the token. For training the model, target labels are taken by fuzzy matching between the token and the KG entity labels.

Kuo and Lu [12] proposed a combined of BiLSTM-CRF model, which takes a joint embedding of question tokens and their POS tags. Here, every token in the question is associated with a label for entity detection problem, namely whether it is an entity, a relation, a question word, etc. These labels are obtained from the SPARQL query that corresponds to the question (as given by the dataset) by matching entities and relations in the query with the question tokens. The model then predicts the most appropriate label sequence that matches the true label sequence associated with the question tokens. Note, however, that this model does not capture the information regarding head and tail position of the entities.

Bakhshi et al. [32] use DBPedia Spotlight to annotate the mentions of resources in DBPedia. The tool is also used to create a dictionary containing a phrase in the NLQ and the corresponding entity in the KG. The dictionary is used by the downstream task, namely entity linking. However, the use of DBPedia tool will constraint to only the entities that appear in DBPedia.

With regards to the issue of undetected entities by NER, Hu et al. [13] borrowed a dictionary-based entity linking proposed by Deng et al. [33] to address the problem of questions without named entity at all positions. Words or phrases mentioned in the question are matched in words or phrases in the dictionary using various similarity functions. Similarly, Bakhshi et al. [14] addressed the problem of an NLQ containing no named entity by implementing a dictionary to store English lexicon and the corresponding resources in a KG. Mentions in the NLQ is matched with an entry in the dictionary. However, since the dictionary is built manually, adapting the approach to other data is not easy.

Cui et al. [15] proposed a pipeline of KGQA systems that consists of entity detection, entity linking, and relation prediction. They proposed a solution to entity detection by training a sequence labeling model based on a combination of BiGRU and CRF. This approach is similar to Kuo and Lu [12] where the aim is to predict the most appropriate label sequence for the question tokens. But unlike Kuo and Lu's work, Cui et al. employed Viterbi algorithm to compute the optimal label sequence, which is rather costly in both memory usage and computation time.

Azmy et al. [16] tackled the problem of entities not explicitly mentioned in an NLQ using a hybrid approach of BiLSTM and CRFs. N-grams and Levenshtein distance are used to exact-match mentions in the NLQ with mentions that are actually considered as entities according to the ground truth. If no matching entity is found in the NLQ, it searches the corresponding tokens mentioned in the NLQ by choosing the longest matching n-gram. However, this approach does not provide the position of entities in the triple patterns of the SPARQL query.

Proposed method

In this section, we present our solution to the entity detection task. In essence, it consists of a transformer model trained to recognize the token-based positions of entities in a given NLQ and whether those entities are head or tail entities.

We use Transformer since the model uses an attention mechanism. Transformer differs from the encoder-decoder architecture based on vanilla Recurrent Neural Networks (RNNs) in that the latter only pays attention to the last encoder state. Transformer

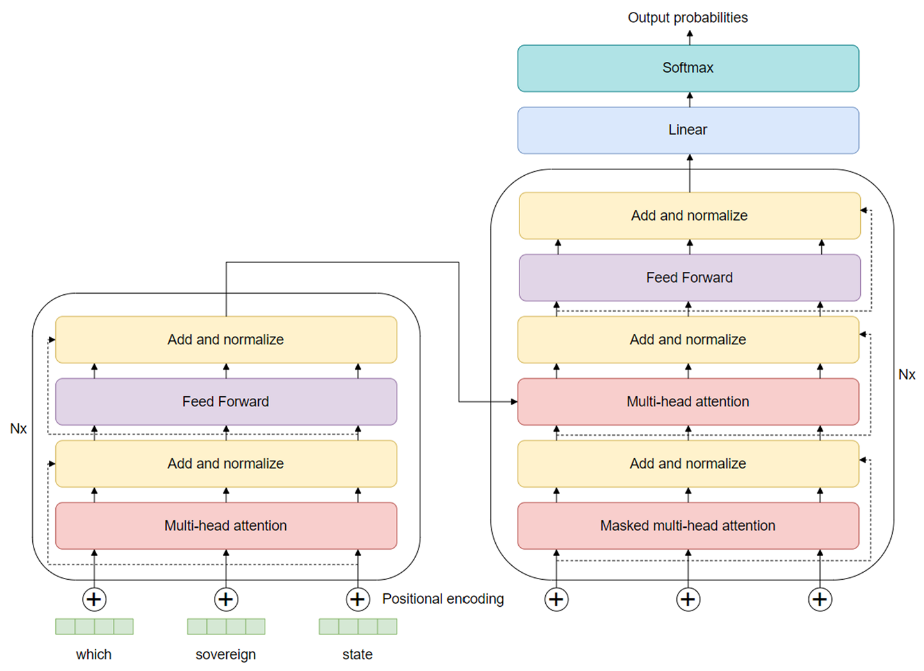


Fig. 3 Transformer model architecture

looks at all the encoder states for each decoder step and computes the most appropriate encoder state to use as the underlying context to generate the decoder's output at that particular step. This allows the model to keep relevant information of the earlier elements of the sequence, which is beneficial for understanding long sequences such as text-based paragraphs. In the context of KGQA, such long sequences could appear in complex questions that are associated with multiple triple patterns in the corresponding queries.

We start with an overview of the Transformer model we use. Next, we describe how we construct the target labels for the model by employing both the training questions and their corresponding ground truth triples. At the end, we describe how we train the model to predict those labels. Meanwhile, our second contribution in which we provide an expansion to the current benchmark QALD datasets is discussed in the Experimental Evaluation section.

Transformer

Transformer is an encoder-decoder network that relies on attention mechanisms rather than recurrence and convolutions [34]. Figure 3 shows the transformer architecture. In its most basic form, a transformer learns to generate an output sequence from a given input sequence. The encoding component works on an embedding of the input sequence through a series of encoder networks, each of which consists of a self-attention layer and a feed-forward neural network. It results in a representation of the input sequence in the form of a set of attention vectors. Meanwhile, the decoding component operates on an embedding of the output sequence via a series of decoder networks. Each decoder also has a self-attention layer and a feed-forward neural network, but with an additional encoder-decoder attention layer in between. This additional layer makes use of the attention vectors from the encoding component to guide the decode to relevant parts of the input. At the end, the decoding component is stacked with a linear and softmax layer that returns a set of probability values for each word in the context of other words in the sentence.

Transformer can be used for a variety of NLP tasks. For text classification, both the input and output to the transformer are actually the same input sentence. Thus, the transformer's weights form a feature representation of the data. The classification is then achieved by adding another classification layer on top of the transformer where the actual target labels of the task come into play. In essence, entity detection in this paper is a text classification task. Our implementation makes use of BERT, a pre-trained transformer obtained from HuggingFace's Transformers library [35] with the help of a wrapper library from SimpleTransformers.[4]

Position-based pattern

We describe in this section the notion of position-based pattern and position-based pattern set as motivated by the Candice Bergen example in the Introduction section. Recall that a position-based pattern needs to contain three types of information: (i) in which triple pattern of the query (first, second, etc.) the entity appears, (ii) whether the entity appear in the head or tail position, and (iii) the token position of the entity in the NLQ. Note that the token position allows us to locate the occurrence of entities in the input NLQ. We use each set of such patterns to represent a particular question type to which the Transformer model learns to classify the questions.

Definition 1 A *triple pattern* is of the form $p = (h, r, t)$ where $h = \texttt{head}(p)$ and $t = \texttt{tail}(p)$ are respectively called the *head* and the *tail* of p, r is the *predicate* of p. All of h, r, and t can be a word or a phrase in a natural language or an entity/relation identifier with respect to a KG. In addition, h and t can be a variable. A triple pattern (h, r, t) is called a *head pattern* if h is not a variable, and a *tail pattern* if t is not a variable. Note that a triple pattern can be both a head and a tail pattern.

Definition 2 We write an NL sentence q consisting of n words/tokens as a list with non-negative integer index $q = [w_0, w_1, \ldots, w_{n-1}]$. A *token-based position* is a list of consecutive non-negative integers.

Given a phrase $s = [v_0, v_1, \ldots, v_{k-1}]$ containing k words/tokens and an NL sentence q, one can associate *POS*(s, q) of token-based position at which s occurs first in q. That is, *POS*(s, q) is a list of the form $[m, m+1, \ldots, m+k-1]$ where m is the smallest index such that $v_0 \equiv w_m, v_1 \equiv w_{m+1}, \ldots, v_{k-1} \equiv w_{m+k-1}$ where $\equiv$ denotes case-insensitive string equality. If s does not occur in q, *POS*(s, q) is undefined.

POS(s, q) provides the leftmost occurrence of s in q. For example, in the question "Where in the United States was John Morris Russell born?", the token-based position of the phrase "John Morris Russell" with respect to the question is the list "[6,7,8]".

[4] https://simpletransformers.ai.

Our aim is to classify a given question according to whether all of its *anchor* entities, i.e., those that are necessary for obtaining a correct answer, potentially occurs in the KG as a head or a tail of a triple. Moreover, questions for which their anchor entities occur at different positions should be distinguished from each other. Thus, different combination of token-based positions with a `head/tail` tag form the different target class labels. Such a `head/tail` tag for a question in the training set can be constructed based on the ground truth triples for that question. Assuming training questions sufficiently represent all types of questions, all those combinations of `head/tail` tag and token-based positions capture any type of unseen questions.

Definition 3 A *position-based pattern* is a pair (x, ℓ) where x is either `head` or `tail` and ℓ is a token-based position.

Let (q, T_q) be a pair of a question q and a non-empty list of triple patterns $T_q = [p_0, p_1, \dots, p_{m-1}]$. We call such a pair a *question-answer pair* and a KGQA dataset can be viewed as a finite set of such pairs. The triple patterns represent the ground truth answer for q and can either be given explicitly as triples or as a part of a logical query (e.g., SPARQL) that gives an answer to q when executed on the KG.

Definition 4 Given a question-answer pair (q, T_q) with respect to a KG $\mathcal{G}$ where $T_q = [p_0, p_1, \dots, p_{m-1}]$, the *position-based pattern set* for q is a set C_q of indexed position-based patterns such that for each $i = 0, 1, \dots, m-1$:

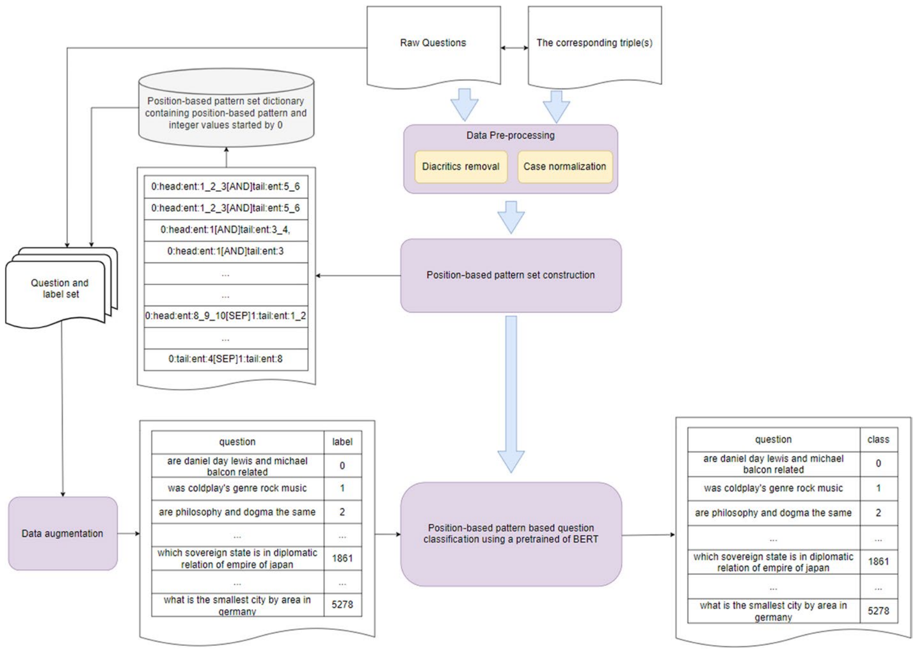
(i) if p_i is a head pattern and $\texttt{head}(p_i)$ has s_i as its NL label in $\mathcal{G}$, then $i{:}(\texttt{head}, POS(s_i, q)) \in C_q$ provided that $POS(s_i, q)$ is defined; and
(ii) if p_i is a tail pattern and $\texttt{tail}(p_i)$ has s_i as its NL label in $\mathcal{G}$, then $i{:}(\texttt{tail}, POS(s_i, q)) \in C_q$ provided that $POS(s_i, q)$ is defined.

Each unique position-based pattern set forms a class label for the question classification task. The `head/tail` tag is based on the ground truth triple patterns, not on the occurrence(s) of s in q. Note that even if s appears more than once in q, we only use the leftmost occurrence of s in q for $POS(s, q)$ because once s at $POS(s, q)$ is linked to a KG entity (in the subsequent step after entity detection), all occurrences of s in q will also be linked to it.

To illustrate the definition of position-based pattern set, consider the question $q =$ "Is Amedeo Maiuri and Ettore Pais excavation directors of Pompeii?" from the LC-QuAD 2.0 dataset. The ground truth triple patterns T_q for q are given inside the following query that can be executed over Wikidata:

```
ASK WHERE  {  wd:Q43332 wdt:P4345 wd:Q442340  .
              wd:Q43332 wdt:P4345 wd:Q981427  .  }
```

Thus, $T_q = [p_0, p_1]$ where $p_0 = (\text{wd:Q43332}, \text{wdt:P4345}, \text{wd:Q442340})$ and $p_1 = (\text{wd:Q43332}, \text{wdt:P4345}, \text{wd:Q981427})$. Note that p_0 and p_1 are both head


Raw Questions
The corresponding triple(s)
Position-based pattern set dictionary containing position-based pattern and integer values started by 0
Data Pre-processing
Diacritics removal
Case normalization
Question and label set
Position-based pattern set construction
Data augmentation
Position-based pattern based question classification using a pretrained of BERT

0:head:ent:1_2_3[AND]tail:ent:5_6
0:head:ent:1_2_3[AND]tail:ent:5_6
0:head:ent:1[AND]tail:ent:3_4,
0:head:ent:1[AND]tail:ent:3
...
...
0:head:ent:8_9_10[SEP]1:tail:ent:1_2
...
0:tail:ent:4[SEP]1:tail:ent:8

question	label
are daniel day lewis and michael balcon related	0
was coldplay's genre rock music	1
are philosophy and dogma the same	2
...	...
which sovereign state is in diplomatic relation of empire of japan	1861
...	...
what is the smallest city by area in germany	5278

question	class
are daniel day lewis and michael balcon related	0
was coldplay's genre rock music	1
are philosophy and dogma the same	2
...	...
which sovereign state is in diplomatic relation of empire of japan	1861
...	...
what is the smallest city by area in germany	5278

Fig. 4 The proposed of entity detection via position-based pattern

and tail patterns. Moreover, "Amedeo Maiuri", "Ettore Pais", and "Pompeii" are labels of wd:Q442340, wd:Q981427, and wd:Q43332 in Wikidata and all of them appear in q. That is, the token-based positions are $POS(\text{Amedeo Maiuri}, q) = [1, 2]$, $POS(\text{Ettore Pais}, [4, 5])$, and $POS(\text{Pompeii}, q) = [9]$. Thus, the class label of the question q is:

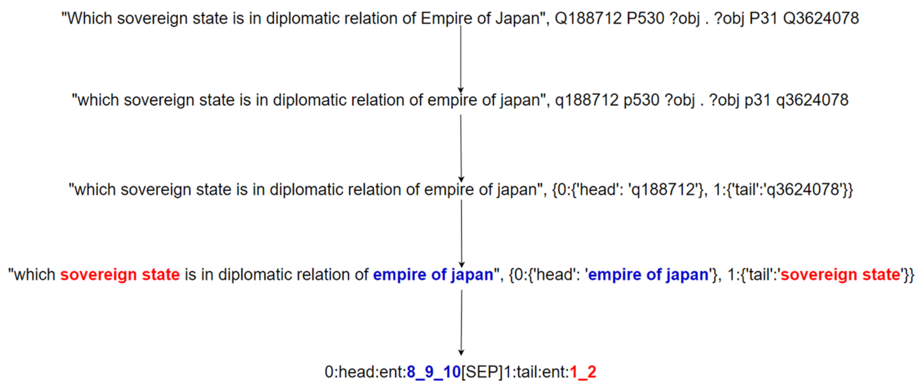
$C_q = \{0{:}(\texttt{head}, [9]), 0{:}(\texttt{tail}, [1, 2]), 1{:}(\texttt{head}, [9]), 1{:}(\texttt{tail}, [4, 5]\}$.

Position-based pattern set construction

Our proposed approach for entity detection based on position-based pattern consists of two main phases, as shown in Fig. 4. The first phase is the position-based pattern set construction, which is discussed in this section. The second phase is the classification of the input question into one of the position-based pattern sets using a transformer-based multiclass classifier, which will be discussed in the subsequent section.

The position-based pattern set construction is described in Fig. 4. The input is a question-answer pair in the dataset. Note that the answer component of the pair is either a SPARQL query or a ground triple without variables. In both cases, we can obtain a set of triple patterns expressing the answer. Next, we perform diacritics removal and case normalization over the input question as a part of data pre-processing step.

Let (q, T_q) be the question-answer pair with q already pre-processed. We construct the position-based pattern set for q by following the steps in Algorithm 1. We illustrate the process via a running example given in Fig. 5.



```
"Which sovereign state is in diplomatic relation of Empire of Japan", Q188712 P530 ?obj . ?obj P31 Q3624078
"which sovereign state is in diplomatic relation of empire of japan", q188712 p530 ?obj . ?obj p31 q3624078
"which sovereign state is in diplomatic relation of empire of japan", {0:{'head': 'q188712'}, 1:{'tail':'q3624078'}}
"which sovereign state is in diplomatic relation of empire of japan", {0:{'head': 'empire of japan'}, 1:{'tail':'sovereign state'}}
0:head:ent:8_9_10[SEP]1:tail:ent:1_2
```

Fig. 5 Running example of position-based pattern set construction

Algorithm 1: Position-based pattern set construction algorithm

```
   Input: $q$ — an NLQ already cased-normalized with tokens separated by a single space,
          $T_q$ — a nonempty zero-based indexed list of triple patterns
   Result: $C_q$ — a position-based pattern set
 1 $ps \leftarrow \{\}$
 2 for $i = 0$ to length$(T_q) - 1$ do
 3     $t \leftarrow T_q[i]$
 4     $s \leftarrow$ head$(t)$
 5     if $s$ is not a variable then
 6         $\ell_s \leftarrow$ KG label or text description of $s$
 7         Add $i$:{head:$\ell_s$} to $ps$
 8     end
 9     $o \leftarrow$ tail$(t)$
10     if $o$ is not a variable then
11         $\ell_o \leftarrow$ KG label or text description of $o$
12         Add $i$:{tail:$\ell_o$} to $ps$
13     end
14 end
15
16 $C_q \leftarrow \{\}$
17 foreach element $p$ in $ps$ do
18     let $p$ be of the form $i$:{$pos$:$\ell$}
19     $N \leftarrow$ number of tokens in $\ell$
20     for $n = N$ to 1 do
21         $matched \leftarrow$ false
22         foreach $n$-gram substring $w$ of $\ell$ do
23             if $w$ is a substring of $q$ then
24                 $tokenpos \leftarrow$ list of token indices where $w$ occurs in $q$
25                 Add $i$:{$pos$:$tokenpos$} to $C_q$
26                 $matched \leftarrow$ true
27                 break
28             end
29         end
30         if $matched$ = true then break
31     end
32 end
```

The first part of the algorithm is described between line 1 and line 14. In this part, we go through all triple patterns in T_q and extract any occurrence of entities either in the head or tail position of the triple pattern. For a triple pattern (s, p, o), s and o can be an entity or a variable. If s is not a variable, then we encounter a head entity, and if o is not a variable, we obtain a tail entity. In both cases, we mark the occurrence by adding i:{pos:ℓ} to the set ps where i is the index of the triple pattern (0th, 1st, etc.), pos is either `head` or `tail` depending on whether we encounter a head or tail entity, and ℓ is a text representation of the entity obtained by querying its label or description in the KG. For example, using an NLQ in Fig. 5, we first obtain “empire of japan” and “sovereign state” as the entity label of ’Q188712’ in the first triple and ’Q3624078’ in the second triple, respectively. By the algorithm (line 7 and 12), we add 0:{`head` : ‘empire of japan’} and 1:{`tail` : ‘sovereign state’} to *ps*.

In the second part of the algorithm, described between line 16 and 32, we perform N-gram matching of ‘empire of japan’ and ‘sovereign state’ in the question string q. That is, we determine whether ‘empire of japan’ and ‘sovereign state’ or any of their n-gram substrings occurs in q. Line 17 starts a loop over elements of *ps*. That is, for the element of *ps* containing the KG label ‘empire of japan’, line 20-30 determines whether ‘empire of japan’ or any of its n-gram substring occurs in q. If so, *tokenpos* stores the token position where it occurs, which is [8, 9, 10] for ‘empire of japan’. When this is found, we add 0:{`head`, $[8, 9, 10]$} to C_q as ‘empire of japan’ is the head entity of the 0th triple pattern. Similarly, in case of ‘sovereign state’, we add 1:{`tail`, $[1, 2]$} to C_q. Thus, in the end, we obtain the resulting $C_q =$ {0:{`head`, $[8, 9, 10]$}, 1:{`tail`, $[1, 2]$}}.

As illustrated in Fig. 4, we store C_q in a dictionary of position-based pattern sets using an ad-hoc syntax, which could ease the downstream task of query construction. The syntax uses the following simple rules:

(1) A list of integers $[x_1, \ldots, x_n]$ is written as `x_1_x_2_..._x_n`. For example, [8, 9, 10] becomes `8_9_10`.
(2) A position-based pattern of the form *i*:{`head`:*tokenpos*} is written in a syntax of the form `i:head:ent:`L where L is the representation of *tokenpos* according to the previous rule. For example, 0:{`head`, $[8, 9, 10]$} is encoded as `0:head:ent:8_9_10`.
(3) A position-based pattern of the form *i*:{`head`:*tokenpos*} is written in a syntax of the form `i:head:ent:`L similar to (2). For example, 1:{`tail`, $[1, 2]$} is encoded as `1:tail:ent:1_2`.
(4) If C_q contains two position-based patterns of the form *i*:{`head`:$tokenpos_h$} and *i*:{`tail`:$tokenpos_t$}, then the two position-based patterns are encoded as a single representation `i:head:ent:`L_h`[AND]i:tail:ent:`L_t where L_h and L_t are representations of $tokenpos_h$ and $tokenpos_t$, respectively according to (1). For example, if C_q contains both 1:{`tail`, $[1, 2]$} and 1:{`head`, $[5, 6]$}, then we have a single encoding `1:head:ent:5_6[AND]1:tail:ent:1_2`.
(5) The representation of C_q is E_1`[SEP]`E_2`[SEP]` ...`[SEP]`E_m where each E_j is the representation of the position-based patterns according to (2) and (3) sorted by their indices. Thus, if $C_q =$ {0:{`head`, $[8, 9, 10]$}, 1:{`tail`, $[1, 2]$}}, then its encoding is `0:head:ent:8_9_10[SEP]1:tail:ent:1_2` as seen in Fig. 5.

Question classification using position-based pattern set

Having constructed position-based pattern sets according to the previous section, we can now use those sets as distinct class labels for our multi-class classification problem. This section describes the step we take to train the model.

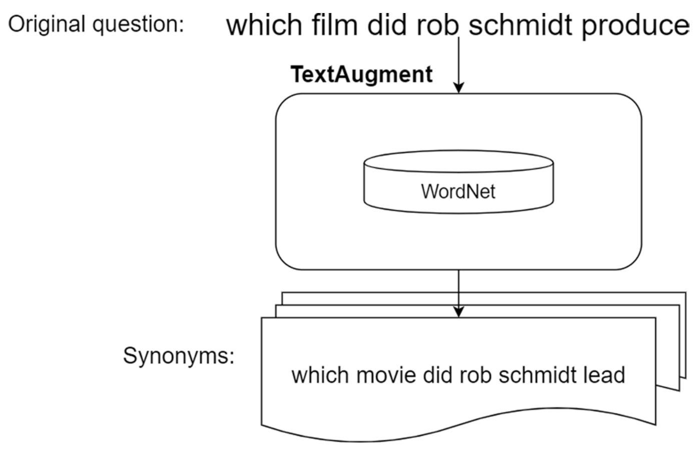

Original question:
which film did rob schmidt produce
TextAugment
WordNet
Synonyms:
which movie did rob schmidt lead

Fig. 6 Synonym based data augmentation

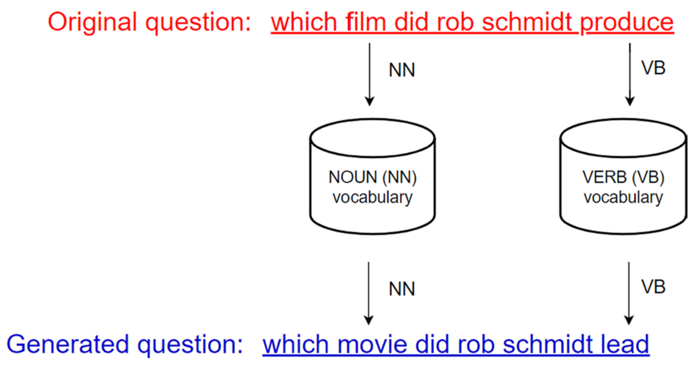

Original question: which film did rob schmidt produce
NN
VB
NOUN (NN) vocabulary
VERB (VB) vocabulary
NN
VB
Generated question: which movie did rob schmidt lead

Fig. 7 Synonym selection

Data augmentation

To improve our model to recognize other words used in a question, we augment data using the synonyms approach. Wordnet is a database containing lexical in English. Each word in the form of nouns, verbs, adjectives, and adverbs are grouped into synonyms (synset).[5] We prefer to use a Wordnet-based approach to preserve the position of words in a question. We borrow Wordnet-based augmentation proposed by Marivate and Sefara [36] to augment data. Figure 6 illustrates how data augmentation proposed by Marivate and Sefara [36] works.

For the augmentation step, we focus on exploring the synonym of verb and noun POS tags in a question. Meanwhile, words with other POS tags are kept as the original. This approach enriches other verb expressions that people use to construct a query. To generate synonym questions that have the same attention as the original question, we replace nouns and verbs that have the same POS tags type; for instance, word "film" and "movie" in Fig. 6 have the same POS tags noun, namely, NN. Word "produce" is replaced by other words with the same POS tags (VB), such as "lead". Figure 7 illustrates how word selection works.

[5] https://www.dbpedia.org.

Position-based pattern set prediction

To predict the position-based patterns of a question, we employ a multi-class classifier with an underlying transformer model. For the transformer model, we use a pre-trained BERT-base model, and on top of it, a classification layer with n output neurons is placed.

As in Fig. 4, Position-based pattern set construction phase outputs a list that contains a pair of questions and their position-based pattern. As in Fig. 5, the pair contains "which sovereign state is in diplomatic relation of empire of japan, 1861". The left (0) and right (1) columns represent a question and ID. of position-based pattern of the question, respectively. In this example, ID 1861 refers to $0 : head : ent : 8\_9\_10[SEP]1 : tail : ent : 1\_2$ pattern in a pattern dictionary. The column we use as the input of this model is the first one.

To predict the position-based pattern set of a question, we use the pre-trained BERT-base-cased model coupled with a simple softmax layer. This model is fine-tuned for a multiclass classification task with our data. The actual implementation of the model is taken from the Simple Transformers[6] library that simplifies the Transformer API from the HuggingFace library. This implementation allows us to train and evaluate the model quickly.

Experimental evaluation

Experiment setup

Machines used

The experiments were conducted in an NVIDIA GPU GeForce GTX with 24 GB total memory for training using GPU driver version 384.130. For data pre-processing, we use 128GB memory. The machine itself is a sever with 3.0 GHz CPU Intel(R) i7-5960X.

Question benchmark datasets

We use 11 different datasets to evaluate our model, namely, SimpleQuestions, LC-QuAD 2.0, and QALD series (there exists nine QALD versions at the moment). However, since QALD series does not provide the Wikidata-based ground truth (exclude QALD-7), we first create Wikidata-based ground truth for all QALD series (except QALD-7). To construct it, we explore entities and relations that appear in the ground truth by manually searching the correspondence entities label into Wikidata. If we do not find any information on Wikidata, we use Google[7] to reveal related information to a keyword that refers to Wikipedia.[8] We use a link that exists on Wikipedia that refers to Wikidata item to find an item mentioned in a question. More details on the construction are explained in the Expansion of QALD benchmark datasets to Wikidata based query section.

In this experiment, we employ the SimpleQuestions dataset for the simple questions as it is the largest dataset for simple questions. The dataset contains 100k questions written by human annotators and associated with Freebase facts [4]. However, since Freebase was terminated on August 31, 2016, and it implies the resource of Freebase is no longer

[6] https://github.com/ThilinaRajapakse/simpletransformers.

[7] https://www.google.com/.

[8] https://www.wikipedia.org.

Table 1 Parameter setup

Argument	Type	Value	Description
pretrained_model	–	bert-base-cased	A small BERT pre-trained model
framework	-	PyTorch	Machine learning framework
num_train_epochs	int	5	The number of epochs
max_seq_length	int	128	Maximum sequence length the model support
train_batch_size	int	8	Training batch size
eval_batch_size	int	8	Evaluation batch size
fp16	bool	False	fp16 mode used
dataloader_num_workers	int	0	Number of worker processed
learning_rate	float	4e−5	Learning rate for training
n_gpu	int	1	Number of GPU used
optimizer	str	"AdamW"	Optimizer used
save_eval_checkpoints	bool	False	Save model for each epcoh

available for free, we use a translated version in Wikidata KG [37]. The dataset provides the corresponding facts of each question. The SimpleQuestions dataset also already provides separate training, validation, and testing sets.

In addition, we use experiment on complex questions from the LC-QuAD 2.0 dataset. The dataset provides 30,000 questions and their paraphrases with varying complexity. The dataset contains 21,258 distinct entities and 1,310 unique relations with ten types of questions such as boolean, dual intentions, facts with qualifiers, and others spread over 22 unique templates. The dataset also provides the SPARQL query corresponding to the questions to answer questions over Wikidata, and DBPedia [6].

Furthermore, we also evaluate our model on benchmark datasets from the QALD series to analyze its performance on non-synthetic questions. Each QALD version presents challenges in solving the questions over related data. We only take QALD challenges using KGs as the data in this experiment.

Knowledge graph

We use a dump of Wikidata for the underlying knowledge graph.[9] The dump contains two billion facts consisting of 49M and 6K items and properties, respectively.

Data and parameter setup

Other than SimpleQuestions, each dataset consists of two-part of data, namely, training and testing data. We randomly separate training data into two folds, namely 80% for training and 20% for validation. Meanwhile, for the SimpleQuestions dataset, we merge training and validation data into a file, and separate it into 80% and 20% for training and validation data, respectively.

Due to the limitation of our machine, we use the pre-trained BERT-base model, which is smaller than the BERT-large model. As noted in the previous section, we make use of the simpler interface from the Simple Transformers library to train and evaluate

[9] "latest-truthy.nt.bz" from https://dumps.wikimedia.org.

Table 2 Query distribution of QALD series and its translation over Wikidata

QALD version	Training				Testing			
	Total	DBP	WD	Caption	Total	DBP	WD	Caption
QALD-1	50	50	47	OOS= 3	50	50	48	OOS= 2
QALD-2	100	93	89	OOS= 4	99	95	86	OOS= 9
QALD-3	100	93	89	OOS= 4	99	95	89	OOS= 6
QALD-4	200	188	185	OOS= 3	50	48	46	OOS= 2
QALD-5	340	326	276	OOS= 10, PQ= 40	59	58	43	OOS= 5, PQ= 10
QALD-6	350	350	313	OOS= 37	100	100	88	OOS= 12
QALD-7*	214	214	99	OOS= 115	42	42	99	-
QALD-8	219	219	205	OOS= 14	41	41	37	OOS= 4
QALD-9	408	408	381	OOS= 27	150	150	139	OOS= 11

DBP DBPedia, *WD* Wikidata, *OOS* out of scope, *PQ* pseudo query. * Wikidata based query is already exist at the QALD version

our model. Table 1 shows parameters used in this work. The rest of the parameters are set according to their default values from the Simple Transformers configuration. simple$_t rans$

Expansion of QALD benchmark datasets to Wikidata based query

Each QALD version consists of two parts of data, namely, training and testing data, respectively. Each QALD version provides challenges to solve over KGs or hybrid data, i.e., a combination of KGs and unstructured data (text) as corpus. Some KGs used in QALD are DBPedia,[10] MusicBrainz,[11] Drugbank,[12] SIDER,[13] and Diseasome.[14] QALD series introduce a different challenge from other datasets as it leverages SPARQL operators in the query instead of simple UNION. This challenge is not exploited in the other datasets [5]. Therefore, we believe that the construction of Wikidata-based QALD series can enrich the challenges of a KGQA benchmark dataset.[15]

In this work, we only translate DBPedia based queries in the QALD series into Wikidata based queries, i.e., not queries to other KGs. Table 2 presents the distribution of Wikidata based query we translate. "Total" column in Table 2 represents the total question of a QALD. "DBP," and "WD" columns display the number of available queries of the dataset (i.e., DBPedia and Wikidata, respectively). Out of scope states that nothing facts are found on Wikidata after exploring some related routes. For example, the question "Which airports does Air China serve?", "Air China" is identified as a head entity of the question. While "dbo:targetAirPort" is used as the property of the question over DBPedia. However, "Air China" item on Wikidata does not have

[10] https://www.dbpedia.org.

[11] https://musicbrainz.org.

[12] http://www.drugbank.ca/.

[13] http://sideeffects.embl.de.

[14] http://wifo5-03.informatik.uni-mannheim.de/diseasome/.

[15] Expansion of QALD benchmark datasets to Wikidata based query can be found at https://github.com/moh-yani/abet-kgqa/tree/main/wikidata_qald_series/.

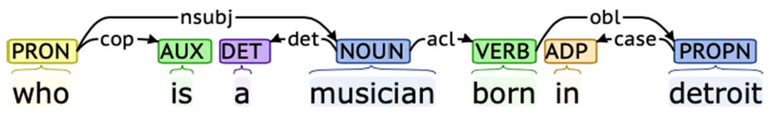


Fig. 8 Example of UDP extraction of a question

any statement that refers to the property. The different data availability on different KGs may happen. Even the original QALD-7 has a significant gap in number between DBPedia based query and Wikidata based query constructed by Usbeck et al. [26], namely 116%. In contrast to Wikidata based query of QALD-7 proposed by Usbeck et al. [26]. The gap of our translation version is lesser, namely 8%. In addition, the question types of untranslated one have been represented by other translated questions. Therefore, our translation version of Wikidata can be reasonable to evaluate a KGQA system that addresses QALD series challenges.

Meanwhile, pseudo queries constructed using unstructured data (text) are excluded from our work because the data is explored through non-KG sources. For instance, a question on QALD-5 [24] "Which anti-apartheid activist was born in Mvezo?" queries information from text resources as below:

```
SELECT DISTINCT ?uri
WHERE {
        ?uri rdf:type text:"anti-apartheid activist"} .
        ?uri dbo:birthPlace res:Mvezo .
}
```

The first triple of the query above uses `text:"anti-apartheid activist"` as a tail entity. However, it does not refer to an item on a KG but a text instead. Since we focus on only query over KGs, we exclude such a query form in our work.

Evaluation method

Our model receives inputs in the form of NL questions and then classifies them into an appropriate position-based pattern set. However, we were not completely convinced that feeding the questions (after pre-processing) to the model in a plain text form is the most appropriate way. To ensure this, we first explore a few different ways of feeding them to the model. Specifically, we consider four ways of feeding an input question to the model, namely (i) in plain text form (as a sequence of words); (ii) as a sequence of features from the corresponding universal dependency tree; (iii) in a hybrid form, mixing case (i) and (ii); and (iv) as a sequence of POS tags.

As an example, consider the question "who is a musician born in detroit", which has been pre-processed as usual. We conduct four experiments, one for each of (i) until (iv) over the Lc-QuAD 2.0 and SimpleQuestions datasets, which represent complex and simple question types. We call experiments for case (i) as text-based feature experiments. Here, the question text is directly fed to the model, yielding a performance listed in Table 3.

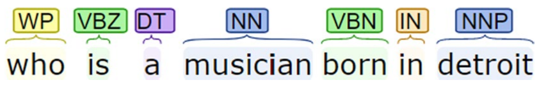


Fig. 9 Example of POS tagging of a question

Case (ii) corresponds to the universal dependency feature experiment. In this experiment, we use the universal dependency parsing (UDP) of the questions as data input. Although the input is also in text-based form, it is not in natural language forms. The UDP of the question example above is as illustrated in Fig. 8. UDP in Fig. 8 is encoded into a text-based graph-like.stanza The text-based graph contains subject, predicate, object, and complements. For instance, the word "musician" is encoded into "id:4 word:NOUN deprel:nsubj id:1 word:PRON". The subject, predicate, object, and complements are represented by "id:4", "word:NOUN", "deprel:nsubj", "id:1" and "word:PRON", respectively. All words of the question are encoded in a text-based graph. So, the complete input of the question is "id:1 word:PRON deprel:root id:0 word:PROPN. id:2 word:AUX deprel:cop id:1 word:PRON. id:3 word:DET deprel:det id:4 word:NOUN. id:4 word:NOUN deprel:nsubj id:1 word:PRON. id:5 word:VERB deprel:acl id:4 word:NOUN. id:6 word:ADP deprel:case id:7 word:PROPN. id:7 word:PROPN deprel:obl id:5 word:VERB". Table 4 shows the performance of this experiment.

Case (iii) corresponds to the hybrid-based feature experiment. The experiment tries to hybrid natural language-based text and non-natural language-based text as the input of our model. From the previous question example, the input is "id:1 word:PRON deprel:root id:0 word:PROPN. id:2 word:AUX deprel:cop id:1 word:PRON. id:3 word:DET deprel:det id:4 word:NOUN. id:4 word:NOUN deprel:nsubj id:1 word:PRON. id:5 word:VERB deprel:acl id:4 word:NOUN. id:6 word:ADP deprel:case id:7 word:PROPN. id:7 word:PROPN deprel:obl id:5 word:VERB who is a musician born in detroit". This experiment yields a performance shown in Table 5.

Case (iv) corresponds to the POS tags-based feature experiment. Here, OS tags are formed in a string format by concatenating POS tags of each token to be a sequence of POS tags. For the question example above, we convert POS tags of the question such as shown in Fig. 9 to be "WP VBZ DT NN VBN IN NNP".stanza. Table 6 shows the performance of this experiment.

Experiment Results

We use testing data to evaluate our work. We match the predicted classes obtained by the classification model with entity ground truth on testing data. For example, the predicted pattern of a question "Which sovereign state is in diplomatic relation of Empire of Japan?" is taken as a correct prediction if both ground truth and its prediction have the same pattern, namely: 0:`head`:`ent`:1_2`[AND]tail`:`ent`:5_6_7. We use a confusion matrix to compute accuracy, precision, recall, and F1. Jurafsky and Martin [38, chapter 4] presented the precision, recall, and F1 as shown in Equation (2), (3), and (4), respectively.

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{1}$$

Table 3 The performance of entity detection task using text-based feature (in percent)

Dataset	#Class	Accuracy	Precision	Recall	F1	Training duration (hour)
SimpleQuestions	1,014	99.15	99.25	99.15	99.17	164
LC-QuAD 2.0	4,296	74.09	75.72	74.09	73.87	211

Table 4 The performance of entity detection task using UDP-based feature (in percent)

Dataset	#Class	Accuracy	Precision	Recall	F1	Training duration (hour)
SimpleQuestions	1,014	43.69	73.31	43.69	49.03	167
LC-QuAD 2.0	4,296	51.18	74.54	51.18	56.43	215

Table 5 The performance of entity detection task using UDP-based feature (in percent)

Dataset	#Class	Accuracy	Precision	Recall	F1	Training duration (hour)
SimpleQuestions	1,014	50.86	78.3	50.86	56.37	164
LC-QuAD 2.0	4,296	52.41	75.49	52.41	57.74	212

Table 6 The performance of entity detection task using POS tagging-based feature (in percent)

Dataset	#Class	Accuracy	Precision	Recall	F1	Training duration (hour)
SimpleQuestions	1,014	98.5	98.5	98.5	98.5	168
LC-QuAD 2.0	4,296	59.6	63.9	59.6	59.1	216

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

$$Recall = \frac{TP}{TP + FN} \tag{3}$$

$$F_1 = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{4}$$

Tables 3, 4, 5, and 6 show the best model obtained by each experiment.

For the measurement, since we want to assign greater contributions to classes that have a bigger number than others, we use a weighted average for the datasets used in this experiment (Tables 3, 4, 5, 6, 7). The tables measure the performance of entity detection tasks only, not the end-to-end performance of a KGQA system. Although there are query ground truth in datasets, we do not check the accuracy of the whole query but only check if the predicted entity matches the entity of ground truth.

```
SELECT DISTINCT ?sbj where {
    ?sbj wdt:P108 wd:Q32491 .
    ?sbj wdt:P31 wd:Q5 }


SELECT DISTINCT ?obj where {
    wd:Q188784 wdt:P461 ?obj .
    ?obj wdt:P31 wd:Q636497
}
```

Fig. 10 Two questions with the same POS tags but different kinds of graph pattern example

According to the experiments, entity detection on SimpleQuestions dataset can be satisfactorily addressed by using both text-based features and POS tagging-based features (Tables 3 and 6). However, entity detection on LC-QuAD 2.0 is hard to solve by POS tagging-based features as those features cannot distinguish two questions with the same POS tags but different subgraphs. For instance, the question "Who is the leader of Qantas" and the question "Who is the opposite of Superhero" have the same sequence of POS tags, namely "WP VBZ DT NN IN NNP". However, although both questions have the same POS tags, they correspond to different kinds of graph patterns. The first question corresponds to a graph pattern containing two triple patterns where the answer variable is located at the head position of both triple patterns (note that there is only a single answer variable as implied by the question). Meanwhile, the second question corresponds to a graph pattern containing two triple patterns where the answer variable is located at the tail position of the first triple pattern and at the head position of the second triple pattern. Figure 10 lists the ground truth query for both questions.

Meanwhile, the first experiment (Table 3) demonstrates a better performance on the entity detection task than the other two for both the SimpleQuestions and LC-QuAD 2.0 datasets. It shows that Transformer works better with plain text-based features. Based on these results, we decided to extend our experiments using only plain text-based features over other datasets, namely, the QALD series. In the fourth experiment, we include QALD series dataset as training data. We merge training data of SimpleQuestions, LC-QuAD 2.0, and QALD series into training data, and separate it into 80% and 20% for training and validation data, respectively. Meanwhile, each testing data of each dataset is used to evaluate our model.

According to Table 7 our model trained and tested on SimpleQuestions outperforms the previous works for about 99.15% of accuracy. Our model trained and tested on LC-QuAD 2.0 is slightly lower than Evseev and Arkhipov [11] in F1 score, but it can be improved by merging with other datasets. Our model achieves 97.4% of accuracy for LC-QuAD 2.0. In contrast to the performance on LC-QuAD 2.0, our model trained in the merged dataset has slightly decreased on SimpleQuestions as it is influenced by the training data of LC-QuAD 2.0 that has a high variation of SPARQL. Our model yields a satisfactory performance on QALD series and outperforms Hu, Zou, Yu et al. [13] and Bakhshi et al. [14] on QALD-6, and QALD-9, respectively. Note that Hu, Zou, Yu et al.

Table 7 Comparison of entity detection performance on testing dataset (in percent). SQ and LQ refer to SimpleQuestions and LC-QuAD 2.0 dataset, respectively

Approach and the dataset(s) used	Acc.	P	R	F1
A. Tested on SQ:				
Dai et al. [7]	75.7	-	-	-
He and Golub [8]	96.8	-	-	-
Chao and Li [9]	82.2	-	-	-
Lukovnikov et al. [10]	82.7	-	-	-
Azmy et al. [16]	-	-	-	90.3
Cui et al. [15]	-	97.4	96.1	96.1
Our approach trained on SQ	99.15	99.25	99.15	99.17
Our approach trained on SQ, LQ, and QALD	97.1	97.9	97.1	97.4
B. Tested on LQ:				
Evseev and Arkhipov [11]	-	-	-	87
Our approach trained on LQ	74.09	75.72	74.09	73.87
Our approach trained on SQ, LQ, and QALD	97.4	98	97.4	97.6
C. Tested on QALD				
Hu, Zou, Yu et al. [13] (QALD 6)	92	–	–	–
Bakhshi et al. [14] (tested on QALD 9)	78	–	–	–
*Kuo and Lu [12] (tested on QALD 7)	91	–	–	–
Our approach: (trained on SQ, LQ, and QALD)				
Tested on QALD-1	95.4	–	–	–
Tested on QALD-2	100	–	–	–
Tested on QALD-3	100	–	–	–
Tested on QALD-4	97.5	–	–	–
Tested on QALD-5	95.1	–	–	–
Tested on QALD-6	96.4	–	–	–
Tested on QALD-7	95.4	–	–	–
Tested on QALD-8	88.2	–	–	–
Tested on QALD-9	96	–	–	–

* Used a rule-based approach, so no training data is used. ** Kuo and Lu's performance is not comparable as they used a different training dataset than ours

Table 8 Model evaluation on entity linking compared with Falcon 2.0 on SimpleQuestions dataset (in percent)

Approach	SimpleQuestions			
	Acc	P	R	F1
Original Falcon 2.0 [19]	–	56	64	60
Original Falcon 2.0 for Uppercase Entities [19]	–	66	75	70
Falcon 2.0 with our approach input	73.9	73.9	73.9	73.8

[13] and Bakhshi et al. [14] used a rule-based approach for the entity detection task. In this experiment, we do not compare with Kuo and Lu [12] as they used different datasets for training.

To prove the effectiveness of our model, we evaluate our model's performance in the subsequent task of KGQA systems, namely, entity linking. We compare our model with Falcon 2.0 [19]. Falcon 2.0 is a tool used to link entities over Wikidata.

In this evaluation, we use the output of our model (list of the prediction of entities mentioned in questions) as the input of Falcon 2.0. Falcon 2.0 then outputs the entities corresponding to the input that exists in a KG, namely, Wikidata. We use testing data of SimpleQuestions dataset for this evaluation.

Table 8 shows the effectiveness of our model if it is used as the feeder input of Falcon 2.0. Compared with the original Falcon 2.0 input, namely, a list of prediction of entities mentioned in questions obtained by Falcon 2.0, overall, our approach can be better in performance of accuracy, precision, recall, and F1, namely, 73.9% and 73.8% for precision and F1, respectively.

Conclusion

This paper presents an approach for entity detection in a question in the context of knowledge graph question answering. Our contribution is on the construction of position-based patterns as target labels for a question classification task. The patterns include the exact position of anchor entities in the question and explicitly indicate whether the entities are a head or a tail entity. Our approach can also determine the number of triples forming a question. It is hoped that this information can better assist the subsequent query construction task.

In addition, this paper has demonstrated four experiments on classification tasks using Transformer, namely, text-based feature, UDP-based feature, hybrid-based feature, and POS tags-based feature. According to the results of the experiments, text-based features reached the best performance on the accuracy, precision, recall, and F1 for both SimpleQuestions, LC-QuAD 2.0, and QALD series. The experiment result demonstrates that our model outperforms the previous works on SimpleQuestions and QALD series. It yields 99.15% of accuracy for SimpleQuestions and 96.15% accuracy on average for QALD series. Meanwhile, our model trained and tested on LC-QuAD 2.0 is slightly lower than Evseev and Arkhipov [11] in F1 score. The experiment also shows that our model trained and tested on a merged dataset (SimpleQuestions, LC-QuAD 2.0, and QALD) outperforms all previous works for SimpleQuestions, LC-QuAD 2.0, and QALD series. Our model achieves 97.4% of accuracy on LC-QuAD 2.0.

Our proposed approach has also been evaluated in Falcon 2.0 to show the effectiveness of our work on the downstream task, namely, entity linking. The result demonstrates that our proposed method can improve the accuracy, precision, recall, and F1 of Falcon 2.0 (an entity and relation linking tool over Wikidata).

Furthermore, this paper presents a Wikidata-based version of QALD series as an expansion of the original DBPedia-based QALD series. For translating the entities, we first search manually for an entity that corresponds to DBPedia entity on Wikidata. If we do not find any proper entities from Wikidata page, we use the Wikidata link from

Wikipedia page to obtain the appropriate entity. This translated version is the first work to the best of our knowledge. Therefore, the dataset will be valuable for other researchers who focus on Wikidata as the KG to evaluate the KGQA systems they developed.

In future work, the result of this work can be used to predict relations in a question. A combination of position-based patterns and the predicted relations can help in addressing the entity ambiguity problem in the entity linking task. Specifically, an appropriate pairing of position-based patterns and predicted relations can reduce a large number of candidate entities to be matched during the linking phase.

Acknowledgements
Not applicable.

Author contributions
Conceptualization, methodology, investigation, software, validation, visualization, writing - original draft: Mohammad Yani; Writing - review & editing, formal analysis, funding acquisition, supervision: Adila Alfa Krisnadhi; Proofreading, final editing, supervision: Indra Budi.

Funding
Supported by Faculty of Computer Science of Universitas Indonesia.

Availability of data and materials
The datasets and models built during the work are available in https://github.com/moh-yani/abet-kgqa.

Declarations

Ethics approval and consent to participate
Not applicatble.

Consent for publication
Not applicable.

Competing interests
The authors declare that they have no competing interests.

Received: 30 December 2021 Accepted: 26 May 2022
Published online: 17 June 2022

References

1. Cyganiak R, Wood D, Lanthaler M. (eds.) RDF 1.1 Concepts and abstract syntax. W3C recommendation, Cambridge, MA, USA (25 February 2014). https://www.w3.org/TR/rdf11-concepts/
2. Hu S, Zou L, Zhang X. A state-transition framework to answer complex questions over knowledge base. In: Riloff E, Chiang D, Hockenmaier J, Tsujii J, editors. Proceedings of the 2018 conference on empirical methods in natural language processing, Brussels, Belgium, October 31– November 4, 2018. Stroudsburg, Pennsylvania: Association for Computational Linguistics; 2018. p. 2098–2108. https://doi.org/10.18653/v1/d18-1234
3. Yani M, Krisnadhi AA. Challenges, techniques, and trends of simple knowledge graph question answering: a survey. Information. 2021;12(7):271. https://doi.org/10.3390/info12070271.
4. Bordes A, Usunier N, Chopra S, Weston J. Large-scale simple question answering with memory networks. CoRR **abs/1506.02075** (2015). 1506.02075
5. Steinmetz N, Sattler K. What is in the KGQA benchmark datasets? Survey on challenges in datasets for question answering on knowledge graphs. J Data Semant. 2021;10(3–4):241–65. https://doi.org/10.1007/s13740-021-00128-9.
6. Dubey M, Banerjee D, Abdelkawi A, Lehmann J. Lc-quad 2.0: a large dataset for complex question answering over wikidata and dbpedia. In: Ghidini C, Hartig O, Maleshkova M, Svátek V, Cruz IF, Hogan A, Song J, Lefrançois M, Gandon F, editors. The semantic web—ISWC 2019—18th international semantic web conference, Auckland, New Zealand, October 26-30, 2019, Proceedings, Part II. Lecture Notes in Computer Science, vol. 11779, pp. 69–78. Berlin/Heidelberg: Springer; 2019. https://doi.org/10.1007/978-3-030-30796-7_5
7. Dai Z, Li L, Xu W. CFO: conditional focused neural question answering with large-scale knowledge bases. In: Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers. Stroudsburg, Pennsylvania: Association for Computational Linguistics; 2016. https://doi.org/10.18653/v1/p16-1076
8. He X, Golub D. Character-level question answering with attention. In: Su J, Carreras X, Duh K, editors. Proceedings of the 2016 conference on empirical methods in natural language processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016. Stroudsburg, Pennsylvania: Association for Computational Linguistics; 2016. p. 1598–1607. https://doi.org/10.18653/v1/d16-1166

9. Chao Z, Li L. The combination of context information to enhance simple question answering. In: 5th international conference on behavioral, economic, and socio-cultural computing, BESC 2018, Kaohsiung, Taiwan, November 12-14, 2018. New York: Institute of Electrical and Electronics Engineers; 2018. p. 109–114. https://doi.org/10.1109/BESC.2018.8697305
10. Lukovnikov D, Fischer A, Lehmann J. Pretrained transformers for simple question answering over knowledge graphs. In: Ghidini C, Hartig O, Maleshkova M, Svátek V, Cruz IF, Hogan A, Song J, Lefrançois M, Gandon F, editors. The semantic web—ISWC 2019—18th international semantic web conference, Auckland, New Zealand, October 26-30, 2019, Proceedings, Part I. Lecture Notes in Computer Science, vol. 11778. Berlin/Heidelberg: Springer; 2019. pp. 470–486. https://doi.org/10.1007/978-3-030-30793-6_27
11. Evseev DA, Arkhipov MY. Sparql query generation for complex question answering with bert and bilstm-based model, vol. 2020-June. ABBYY PRODUCTION LLC, Milpitas, CA 95035; 2020. p. 270–282. https://doi.org/10.28995/2075-7182-2020-19-270-282
12. Kuo CY, Lu EJL. A bilstm-crf entity type tagger for question answering system. New York: Institute of Electrical and Electronics Engineers; 2021. p. 161–166. https://doi.org/10.1109/IAICT52856.2021.9532562
13. Hu S, Zou L, Yu JX, Wang H, Zhao D. Answering natural language questions by subgraph matching over knowledge graphs. Trans Knowl Data Eng. 2018;30(5):824–37. https://doi.org/10.1109/TKDE.2017.2766634.
14. Bakhshi M, Nematbakhsh M, Mohsenzadeh M, Rahmani AM. Sparseqa: sequential word reordering and parsing for answering complex natural language questions over knowledge graphs. Knowl Based Syst. 2022;235:107626. https://doi.org/10.1016/j.knosys.2021.107626.
15. Cui H, Peng T, Feng L, Bao T, Liu L. Simple question answering over knowledge graph enhanced by question pattern classification. Knowl Inf Syst. 2021;63(10):2741–61. https://doi.org/10.1007/s10115-021-01609-w.
16. Azmy M, Shi P, Lin J, Ilyas I.F. Farewell freebase: migrating the simplequestions dataset to dbpedia. In: Bender EM, Derczynski L, Isabelle P, editors. Proceedings of the 27th international conference on computational linguistics, COLING 2018, Santa Fe, New Mexico, USA, August 20-26, 2018. Stroudsburg, Pennsylvania: Association for Computational Linguistics; 2018. p. 2093–2103. https://aclanthology.org/C18-1178/
17. Höffner K, Walter S, Marx E, Usbeck R, Lehmann J, Ngomo AN. Survey on challenges of question answering in the semantic web. Semant Web. 2017;8(6):895–920. https://doi.org/10.3233/SW-160247.
18. Türe F, Jojic O. No need to pay attention: Simple recurrent neural networks work! In: Palmer M, Hwa R, Riedel S, editors. Proceedings of the 2017 conference on empirical methods in natural language processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017. Stroudsburg, Pennsylvania: Association for Computational Linguistics; 2017. p. 2866–2872. https://doi.org/10.18653/v1/d17-1307
19. Sakor A, Singh K, Patel A, Vidal M. Falcon 2.0: An entity and relation linking tool over wikidata. In: d'Aquin M, Dietze S, Hauff C, Curry E, Cudré-Mauroux P, editors. CIKM '20: the 29th ACM international conference on information and knowledge management, virtual event, Ireland, October 19-23, 2020. New York: ACM; 2020. p. 3141–3148. https://doi.org/10.1145/3340531.3412777
20. Unger C. Question answering over linked data: QALD-1 open challenge, 1–11. 2011.
21. Unger C, Cimiano P, Lopez V, Motta E, Buitelaar P. R.C.: QALD-2 Open Challenge. 2012
22. Cabrio E, Cimiano P, López V, Ngomo AN, Unger C, Walter S. QALD-3: multilingual question answering over linked data. In: Forner P, Navigli R, Tufis D, Ferro N, editors. Working notes for CLEF 2013 conference , Valencia, Spain, September 23-26, 2013. CEUR Workshop Proceedings, vol. 1179. London: CEUR-WS.org; 2013. http://ceur-ws.org/Vol-1179/CLEF2013wn-QALD3-CabrioEt2013.pdf
23. Unger C, Forascu C, López V, Ngomo AN, Cabrio E, Cimiano P, Walter S. Question answering over linked data (QALD-4). In: Cappellato L, Ferro N, Halvey M, Kraaij W, editors. Working notes for CLEF 2014 conference, Sheffield, UK, September 15-18, 2014. CEUR Workshop Proceedings, vol. 1180. London: CEUR-WS.org; 2014. p. 1172–1180. http://ceur-ws.org/Vol-1180/CLEF2014wn-QA-UngerEt2014.pdf
24. Unger C, Forascu C, López V, Ngomo A.N, Cabrio E, Cimiano P, Walter S. Question answering over linked data (QALD-5). In: Cappellato L, Ferro N, Jones GJF, SanJuan E, editors. Working notes of CLEF 2015 - conference and labs of the evaluation forum, Toulouse, France, September 8-11, 2015. CEUR Workshop Proceedings, vol. 1391. London: CEUR-WS.org; 2015. http://ceur-ws.org/Vol-1391/173-CR.pdf
25. Unger C, Ngomo A.N, Cabrio E. 6th open challenge on question answering over linked data (QALD-6). In: Sack H, Dietze S, Tordai A, Lange C, editors. Semantic web challenges—third SemWebEval challenge at ESWC 2016, Heraklion, Crete, Greece, May 29–June 2, 2016, Revised Selected Papers. Communications in Computer and Information Science, vol. 641. Berlin/Heidelberg: Springer; 2016. pp. 171–177. https://doi.org/10.1007/978-3-319-46565-4_13
26. Usbeck R, Ngomo A.N, Haarmann B, Krithara A, Röder M, Napolitano G. 7th open challenge on question answering over linked data (QALD-7). In: Dragoni M, Solanki M, Blomqvist E, editors. Semantic web challenges—4th SemWebEval challenge at ESWC 2017, Portoroz, Slovenia, May 28—June 1, 2017, Revised Selected Papers. Communications in Computer and Information Science, vol. 769. Springer; 2017. p. 59–69. https://doi.org/10.1007/978-3-319-69146-6_6
27. Usbeck R, Ngomo A.N, Conrads F, Röder M, Napolitano G. 8th challenge on question answering over linked data (QALD-8) (invited paper). In: Choi K, Anke LE, Declerck T, Gromann D, Kim J, Ngomo AN, Saleem M, Usbeck R, editors. Joint proceedings of the 4th workshop on semantic deep learning (SemDeep-4) and NLIWoD4: natural language interfaces for the web of data (NLIWOD-4) and 9th question answering over linked data challenge (QALD-9) co-located with 17th international semantic web conference (ISWC 2018), Monterey, California, United States of America, October 8th–9th, 2018. CEUR Workshop Proceedings, vol. 2241. London: CEUR-WS.org; 2018. p. 51–57. http://ceur-ws.org/Vol-2241/paper-05.pdf
28. Usbeck R, Gusmita R.H, Ngomo A.N, Saleem M. 9th challenge on question answering over linked data (QALD-9) (invited paper). In: Choi K, Anke LE, Declerck T, Gromann D, Kim J, Ngomo AN, Saleem M, Usbeck R, editors. Joint proceedings of the 4th workshop on semantic deep learning (SemDeep-4) and NLIWoD4: natural language interfaces for the web of data (NLIWOD-4) and 9th question answering over linked data challenge (QALD-9) co-located with 17th international semantic web conference (ISWC 2018), Monterey, California, United States of America, October

8th–9th, 2018. CEUR Workshop Proceedings, vol. 2241. London: CEUR-WS.org; 2018. p. 58–64. http://ceur-ws.org/Vol-2241/paper-06.pdf
29. Song S, Huang W, Sun Y. Semantic query graph based SPARQL generation from natural language questions. Clust Comput. 2019;22(Suppl 1):847–58. https://doi.org/10.1007/s10586-017-1332-3.
30. Zou L, Huang R, Wang H, Yu J.X, He W, Zhao D. Natural language question answering over RDF: a graph data driven approach. In: Dyreson CE, Li F, Özsu MT, editors, International conference on management of data, SIGMOD 2014, Snowbird, UT, USA, June 22–27, 2014. New York: ACM; 2014. p. 313–324. https://doi.org/10.1145/2588555.2610525
31. Huang X, Zhang J, Li D, Li P. Knowledge graph embedding based question answering. In: Culpepper JS, Moffat A, Bennett PN, Lerman K, editors. Proceedings of the twelfth ACM international conference on web search and data mining, WSDM 2019, Melbourne, VIC, Australia, February 11–15, 2019. New York: ACM; 2019. p. 105–113. https://doi.org/10.1145/3289600.3290956
32. Bakhshi M, Nematbakhsh M, Mohsenzadeh M, Rahmani AM. Data-driven construction of SPARQL queries by approximate question graph alignment in question answering over knowledge graphs. Expert Syst Appl. 2020;146: 113205. https://doi.org/10.1016/j.eswa.2020.113205.
33. Deng D, Li G, Feng J, Duan Y, Gong Z. A unified framework for approximate dictionary-based entity extraction. VLDB J. 2015;24(1):143–67. https://doi.org/10.1007/s00778-014-0367-9.
34. Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez A.N, Kaiser L, Polosukhin I. Attention is all you need. In: Guyon I, von Luxburg U, Bengio S, Wallach HM, Fergus R, Vishwanathan SVN, Garnett R, editors. Advances in neural information processing systems 30: annual conference on neural information processing systems 2017, December 4–9, 2017, Long Beach, CA; 2017. p. 5998–6008
35. Wolf T, Debut L, Sanh V, Chaumond J, Delangue C, Moi A, Cistac P, Rault T, Louf R, Funtowicz M, Davison J, Shleifer S, von Platen P, Ma C, Jernite Y, Plu J, Xu C, Scao T.L, Gugger S, Drame M, Lhoest Q, Rush AM. Transformers: state-of-the-art natural language processing. In: Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations. Association for Computational Linguistics, Online; 2020. p. 38–45. https://www.aclweb.org/anthology/2020.emnlp-demos.6
36. Marivate V, Sefara T. Improving short text classification through global augmentation methods. In: Holzinger A, Kieseberg P, Tjoa AM, Weippl ER, editors. Machine learning and knowledge extraction - 4th IFIP TC 5, TC 12, WG 8.4, WG 8.9, WG 12.9 international cross-domain conference, CD-MAKE 2020, Dublin, Ireland, August 25–28, 2020, proceedings. Lecture Notes in Computer Science, vol. 12279. Berlin/Heidelberg: Springer; 2020. p. 385–399. https://doi.org/10.1007/978-3-030-57321-8_21
37. Diefenbach D, Tanon T.P, Singh KD, Maret P. Question answering benchmarks for wikidata. In: Nikitina N, Song D, Fokoue A, Haase P, editors. Proceedings of the ISWC 2017 posters & demonstrations and industry tracks co-located with 16th international semantic web conference (ISWC 2017), Vienna, Austria, October 23rd to 25th, 2017. CEUR Workshop Proceedings, vol. 1963. CEUR-WS.org, London; 2017. http://ceur-ws.org/Vol-1963/paper555.pdf
38. Jurafsky D, Martin JH. Speech and language processing: an introduction to natural language processing, computational linguistics, and speech recognition, 2nd edition. Prentice Hall series in artificial intelligence. London: Prentice Hall, Pearson Education International; 2009. https://www.worldcat.org/oclc/315913020

Publisher's Note